UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P O Box 1450
Alexandria, Virginia 22313-1450
www uspto gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/911,663	07/24/2001	John Edward Ciolfi	MWS-072	3836

959 7590 03/15/2010
LAHIVE & COCKFIELD, LLP
FLOOR 30, SUITE 3000
ONE POST OFFICE SQUARE
BOSTON, MA 02109

EXAMINER	
PILLAI, NAMITHA	

ART UNIT	PAPER NUMBER
2173	

MAIL DATE	DELIVERY MODE
03/15/2010	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE BOARD OF PATENT APPEALS
AND INTERFERENCES

Ex parte JOHN EDWARD CIOLFI

Appeal 2009-002161
Application 09/911,663[1]
Technology Center 2100

Decided: March 15, 2010

Before LEE E. BARRETT, JEAN R. HOMERE, and STEPHEN C. SIU, *Administrative Patent Judges*.

BARRETT, *Administrative Patent Judge*.

DECISION ON APPEAL

This is a decision on appeal under 35 U.S.C. § 134(a) from the final rejection of claims 33-44 and 46. Claims 1-32 and 45 have been canceled. We have jurisdiction pursuant to 35 U.S.C. § 6(b).

We affirm.

---

[1] Filed July 24, 2001, titled "Handling Parameters in Block Diagram Modeling." The real party in interest is The MathWorks, Inc.

STATEMENT OF THE CASE

The invention

The invention relates to processing graphical block diagram parameter expressions. The parameter processing mechanism pools "like non-interfaced parameter expressions," allowing reuse of both uniform and non-uniform data across constant block parameters in the generated code and during model execution. *See* Abstract. The invention is described in greater detail in the findings of fact section, *infra*.

Illustrative claim

Claim 33 is reproduced below:

33. A method of mapping graphical block diagram block parameters in a graphical block diagram modeling environment, comprising:

receiving a plurality of user-defined block parameters;

processing the plurality of user-defined block parameters to produce a plurality of run-time block parameters;

pooling together like non-interfaced run-time block parameters to reuse data for the like non-interfaced run-time block parameters.

The reference

Dunlavey US 6,937,257 B1 Aug. 30, 2005
(based on a provisional application filed Jan. 31, 2001)

The rejection

Claims 33-44 and 46 stand rejected under 35 U.S.C. § 102(e) as anticipated by Dunlavey.

The Examiner's Answer repeats the rejection of claims 16-24, 26-32, and 45 under 35 U.S.C. § 103(a) as unpatentable over Dunlavey and Kodosky. Appellant notes that an amendment canceling claims 16-24, 26-32, and 45 was filed with the original Brief on June 8, 2007, and requests that the rejection of the canceled claims be withdrawn. Br. 4.[2] The rejection of canceled claims is considered withdrawn.

FINDINGS OF FACT

The invention

Appellant's Brief (Br. 3-5) provides a good summary of the invention, which is helpful in understanding the claimed invention.

The claimed subject matter relates to a computer-based graphical environment for creating, modifying, executing (simulating), and/or generating code for models. The elements of a model may be presented as "blocks," having inputs, outputs, and/or parameters ("block parameters") that may influence the behavior of the blocks. When a block receives data, it may perform operations on that data. Such operations are sometimes referred to as "block methods" that are associated with a block. A user may specify the parameters that are to be used by the block using a dialog box.

---

[2] Pages refer to the Supplemental Appeal Brief filed October 12, 2007.

The "user-specified parameters" may be in the form of numerical values, variables defined as constants, interfaced variables, or some combination thereof. An "interfaced variable" is a variable whose value can be changed either during simulation or in generated code.

Models represented as block diagrams may be executed in an interpreted environment or they may be compiled. In addition, code can be generated from the block diagrams.

The claimed subject matter is directed to receiving a model and processing user-specified parameters for one or more blocks in order to convert them to "run-time parameters." The run-time parameters are the block parameters that are used during execution of the block diagram model or in generated code. They are derived from user-specified parameters. The run-time block parameters may be produced from the user-defined block parameters in such a way as to improve or optimize resource allocation, such as the use of memory space during the execution of the model.

Some user-defined block parameters may be processed in such a way that the data for the corresponding run-time block parameters is reused across two or more parameters from one or more blocks. In particular, "like non-interfaced parameters" may be "pooled" together - that is, each run-time parameter is examined during the processing of blocks and a single common representation may be used for multiple parameters. Such a single representation may be, for example, an entry in a run-time parameter table. Spec. 13, ll. 13-31; Figure 12. Parameter pooling only applies to constant run-time parameters, i.e., run-time parameters explicitly declared as constant

or run-time parameters that do not map back to an interfaced variable. Spec. 13, ll. 20-23; Spec. 18, ll. 21-25.

Generally, the parameter pooling process identifies block parameters having parameter data that matches a given criterion, and allocates only one copy of the parameter data for all references to a given parameter data set. That is, one memory location is created for the run-time parameter reference, which memory location is referred to as a "pooled parameter." Spec. 18, l. 26 to Spec. 19, l. 3. "The uniform parameter pooling looks for all registered run-time parameters with the same value and other attributes (i.e., data type, dimension, and so forth) and combines them into one run-time parameter in one location." Spec. 19, ll. 3-5.

Dunlavey

Dunlavey relates to systems and methods for enabling generation of computational models of drug behavior. Col. 1, ll. 11-14. In particular, Dunlavey describes that a common error in drug model construction is a "units" error where the units entered not internally consistent. Col. 2, ll. 20-33. One aspect of the invention is maintaining consistent unit relationships in a graphical pharmacological computational model editor. Col. 2, ll. 62-64.

Dunlavey describes that the user enters units-specifying data for the objects (blocks) using unit expressions and the units-specifying data is translated into multidimensional unit type data. "This multidimensional unit type data is then propagated for each of the statements to identify

inconsistent units while the model is being constructed, and warning messages are displayed when inconsistent units are found." Col. 3, ll. 8-11. "Every numeric variable and constant (i.e. every numeric tern) is defined by this single multi-dimensional unit type, and these unit types are propagated and tracked in every expression." Col. 16, ll. 40-43. "In one embodiment, there are five basic dimensions of physical units: (1) volume, (2) weight, (3) time, (4) amount, and (5) age." Col. 16, ll. 52-54.

Model blocks are translated into equations using an internal parse tree structure and the primitive operations in Figure 4. Col. 17, ll. 63-65.

Dunlavey depicts the process of providing a model in Figure 5. Users may perform a model construction action, such as placing a block or connecting blocks. Col. 18, ll. 44-46. Block ports (variables) are encoded in an internal parse tree as a 16-bit block number and a 16-bit port (variable) number. Col. 18, ll. 51-57. "A block number of -1 is used to signify global variables such as the time variable T." Col. 18, ll. 57-58. When the user selects a compile option, a code generation module translates from drug model blocks into an internal parse tree data structure, then translates to visible syntax language, and then into Fortran source code and the code is located in a single subroutine. Col. 19, ll. 42-60. Variable names are generated using the unique block numbers, col. 19, ll. 60-63; since global variables have the same block number, we find that they would have the same variable name. The high-level language source code is a Fortran subroutine which is then compiled. Col. 22, ll. 18-23.

ISSUE

As argued, the issue is: Has Appellant shown that the Examiner erred in finding that Dunlavey teaches "pooling together like non-interfaced run-time block parameters to reuse data for the like non-interfaced run-time block parameters" as recited in independent claims 33 and 46? In particular, Appellant argues that Dunlavey does not describe "pooling together like non-interfaced run-time block parameters" or the function "to reuse data for the like non-interfaced run-time block parameters."

The patentability of the dependent claims is not separately argued and thus, the rejection of the dependent claims stands or falls with the rejection of independent claim 33.

PRINCIPLE OF LAW

"Anticipation requires the presence in a single prior art disclosure of all elements of a claimed invention arranged as in the claim." *Connell v. Sears, Roebuck & Co.*, 722 F.2d 1542, 1548 (Fed. Cir. 1983).

ANALYSIS

Appellant's principal Appeal Brief fully addresses the Examiner's rejection prior to the Examiner's Answer. The Response to Argument section of the Examiner's Answer provides additional reasoning. Appellant's Reply Brief identifies and addresses five statements by the Examiner in the Examiner's Answer as to where Dunlavey allegedly teaches "pooling together like non-interfaced run-time block parameters to reuse data for the like non-interfaced run-time block parameters." We find that Appellant has

identified all the relevant reasons by the Examiner and we generally agree with Appellant's arguments regarding the Examiner's statement of the rejection. Nevertheless, we refer to the Reply Brief because it addresses the Examiner's most complete statement of the rejection.

The Examiner's rejection would have been clearer if it had taken a specific example in Dunlavey and explained what corresponds to the "like non-interfaced run-time block parameters" and how Dunlavey performs the step of "pooling together like non-interfaced run-time block parameters." There is no dispute that Dunlavey has "user-defined block parameters" and "run-time block parameters" produced by processing the user-defined block parameters. Thus, the Examiner's discussion of run-time parameters (Ans. 10-11) is not disputed. Nor does there seem to be any dispute that Dunlavey inherently has "like non-interfaced run-time block parameters," i.e., that several of the blocks in Figure 2A may use the same constant parameters ("like non-interfaced run-time block parameters"). What the rejection fails to explain is how these constants are "pooled."

As disclosed, each block of a model has one or more parameters ("block parameters") that influence the behavior of the blocks, e.g., the parameters may be part of an equation for that block. Several blocks may use the same parameter, i.e., the parameter has the same value, data type, and so on: these are "like" parameters. The parameters may be constants or variables. The term "non-interfaced run-time block parameters" refers to parameters that do not change during a simulation, e.g., a constant parameter. "Pooling" means that each run-time parameter is examined

during the processing of blocks and only one copy of the parameter data is allocated for all references to a given parameter data set. That is, one memory location is created for the run-time parameter reference, which memory location is referred to as a "pooled parameter." Spec. 18, l. 26 to Spec. 19, l. 3. "The uniform parameter pooling looks for all registered run-time parameters with the same value and other attributes (i.e., data type, dimension, and so forth) and combines them into one run-time parameter in one location." Spec. 19, ll. 3-5. As we understand the invention, if a constant "non-interfaced run-time parameter," such as X, appears as a parameter in several blocks, a single memory location would be used for all parameter references to X; this is a "pooled parameter." While, perhaps, Appellant contemplates that the invention would look to pool parameters with different names as long as they refer to the same thing, e.g., X and Y, the claims do not require pooling parameters with different names.

We consider each of the Examiner's five rationales as argued by Appellant.

First, the Examiner finds that "[t]he multidimensional data type is one example of a variable that is a combination of pooled together like run time block parameters." Ans. 11.

Appellant argues that the multidimensional data type is not used to represent run-time block parameters or to pool together run-time parameters that are used in the model execution time. Reply Br. 6.

Dunlavey describes that the user enters units-specifying data for the objects (blocks) using unit expressions and the units-specifying data is

translated into multidimensional unit type data. The "multidimensional data type" is a data type associated with a parameter that specifies units for a parameter (e.g., liters). The multidimensional data type only ensures that variables and constants have consistent units and is not anything like "pooling together like non-interfaced run-time block parameters." This rationale by the Examiner is not persuasive.

Second, the Examiner finds: "The parameters provided in Figure 4 represent a variable used in code and include descriptions associated with the variables. The 'SetDiscrete' variable includes multiple variables that are grouped together and are jointly distributed where multiple variables are represented in the one variable 'SetDiscrete.'" Ans. 11.

Appellant argues that the SetDiscrete primitive does not pool like run-time parameters. Reply Br. 7. It is argued that "grouping" parameters does not imply that the parameters are "like" parameters as claimed. *Id.* at 8. It is further argued that variables in the group set using the SetDiscrete primitive do not appear in the high-level language source code and cannot be construed as run-time parameters. *Id.* at 9.

Dunlavey describes that the primitives in Figure 4 are "for an internal parse tree data structure for use in translating model blocks into equations" (col. 17, ll. 63-65). The "SetDiscrete" primitive is described in Figure 4 as "used to set a group of categorical variables that are jointly distributed" but is not further described in the specification. It is unclear what is meant by the description and the Examiner does not explain how "setting" a group of

variables meets the limitation of "pooling" parameters. This rationale by the Examiner is not persuasive.

Third, the Examiner finds: "There are other means for pooling together like parameters where a single common representation is used for multiple parameters. For example, categorical distribution block and multivariate distribution block both represent a single representation that is used to pool together multiple variables across time. See column 8, lines 5-17." Ans. 11, 12.

Appellant argues that a categorical distribution block represents a quantity that is known to be variable over time and a multivariate distribution block represents multiple quantities that are known to be variable over time, but these do not pool "non-interfaced" run-time block parameters, i.e., parameters that are not variable during execution. Reply Br. 10.

The Examiner does not explain, and we fail to see how a categorical distribution or a multivariate distribution requires any kind of "pooling" of variables. There is no apparent connection between distributions and combining several run-time parameters into one run-time parameter in one location as required by "pooling." This rationale by the Examiner is not persuasive.

Fourth, as to the limitation "to reuse data for the like non-interfaced run-time block parameters," the Examiner finds:

> Dunlavey does disclose providing means for the user to change values associated with variables where this involves reuse of variables that are in the graphical model including pooled block variables which are

> representations of multiple variables. Based on the user adjustments to variables the same parameters can be used to calculate new results. See column 25, lines 40-51.

Ans. 12-13.

Appellant argues that the adjustment of variables does not anticipate pooling to reuse data for the like non-interfaced block parameters because the adjustment of variables does not involve reuse of data for the pooled parameters. Reply Br. 11-12.

The limitation "to reuse data for the like non-interfaced run-time block parameters" describes a result of pooling parameters. The Examiner has not shown that Dunlavey describes pooling. The Examiner's statement refers to "reuse of variables" by allowing the user to change values, but the limitation at issue is "reuse data for . . . parameters." That is, several parameters are pooled so that they share the same data. Allowing the user to change the data associated with a variable has nothing to do with pooling parameters. This rationale by the Examiner is not persuasive.

<u>Fifth</u>, as to the limitation "to reuse data for the like non-interfaced run-time block parameters," the Examiner finds:

> Furthermore there are representations of variables that are pooled together, the variables being non interface variables. These include global variables which have one instance but can be retrieved and reused multiple times within a procedure. The purpose of the global variable is to provide one definition, with the variable being accessible to multiple procedures and therefore reused multiple times. See column 10, lines 5-15.

Ans. 13.

Appellant argues that a global variable does not reuse data for pooled parameters. It is argued that "[e]ven if the purpose of the global variable were to be similar to the purpose behind pooling like non-interfaced run-time parameters, *arguendo*, it would in no way suggest that the former anticipates the latter." Reply Br. 13.

A global variable appears to meet the claim limitation of "pooling together like non-interfaced run-time block parameters to reuse data for the like non-interfaced run-time block parameters." However, more explanation is required than is provided by the Examiner. Each global variable, say X, in one or more blocks refers to a single representation or data element in memory. For example, each reference to the global variable X in different blocks has the same variable name and refers to the same instance in memory; therefore, the multiple references to global variable X would be "pooled" and the data would be "reused" because all references to variable X would refer to the same data in memory. Global variables can be set by users in the procedure blocks in Dunlavey, col. 10, ll. 5-10, so these are "user-defined block parameters." For these reasons, we find that Dunlavey teaches "pooling together like non-interfaced run-time block parameters to reuse data for the like non-interfaced run-time block parameters."

In addition, it appears that the claims are directed to what is normally done when programs having variables are compiled. As we understand Dunlavey, the equations in the separate blocks (502, Fig. 5) are combined and converted to a program in a high-level language (516, Fig. 5; col. 19, ll. 42-65) and this source code is compiled and linked (518, Fig. 5; col. 22,

ll. 18-23). "In this embodiment, all of the variables are global variables." Col. 22, l. 24. Thus, the variable X occurring in different blocks is found throughout the final program. When programs are compiled, as was well known to those of ordinary skill in the compiler art, each variable in the program is represented by a single memory address. Thus, it seems that all references to variable X from the various blocks in the compiled program are "pooled" because all references refer to a single memory location. This reasoning is in addition to the reasoning about global variables.

CONCLUSION

Appellant has not shown that the Examiner erred in finding that Dunlavey teaches "pooling together like non-interfaced run-time block parameters to reuse data for the like non-interfaced run-time block parameters" as recited in independent claims 33 and 46. The rejection of independent claims 33 and 46, and dependent claims 34-44, under 35 U.S.C. § 102(e) is affirmed.

Requests for extensions of time are governed by 37 C.F.R. § 1.136(b). *See* 37 C.F.R. § 41.50(f).

AFFIRMED

rwk

LAHIVE & COCKFIELD, LLP
FLOOR 30, SUITE 3000
ONE POST OFFICE SQUARE
BOSTON, MA 02109